Theoretical Computer Science 21 (1982) TCS 145–161
North-Holland Publishing Company

SIMULATIONS AMONG MULTIDIMENSIONAL TURING MACHINES

Michael C. LOUI*

Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Communicated by A. Schönhage
Received September 1980
Revised June 1981

Abstract. For all $d \geq 1$ and all $e > d$, every deterministic multihead e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line by a deterministic multihead d-dimensional Turing machine in time $O(T(n)^{1+1/d-1/e}(\log T(n))^{O(1)})$. This simulation almost achieves the known lower bound $\Omega(T(n)^{1+1/d-1/e})$ on the time required. The simulation is interpreted in terms of dynamic embeddings among arrays with local access.

1. Introduction

It is important to understand how efficiently one data structure or machine simulates another. Frequently, a programmer uses a data structure that must be simulated by the natural data structures provided by the programming language. More generally, for conceptual clarity, designers of computers and programs devise levels of "virtual machines". The constructs of abstract machines are defined in terms of operations of simpler machines. For instance, an ALGOL statement is translated into machine code for a PDP-10 computer, and each PDP-10 machine instruction is executed by running several microcode instructions.

Several researchers have investigated static embeddings among data structures [7, 12, 13, 14]. They fix a correspondence between locations in the guest structure and locations in the host structure and study the access costs. For example, Reischuk [11] devised embeddings of multidimensional arrays into complete binary trees that permit rapid on-line simulations of multidimensional Turing machines by machines with tree-structured storage.

We adopt a dynamic viewpoint. Rather than fixing a correspondence, the simulator should determine the representation itself on-line, designating representatives in the host only for cells of the guest that are actually used. Exploiting this idea,

* Supported by the National Science Foundation under Grant No. MCS-77-19754 by the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under Contract N00014-79-C0424 and by the Fannie and John Hertz Foundation. Present address: Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801, U.S.A.

0304-3975/82/0000–0000/$02.75

we develop new techniques for efficient simulation. In particular, we present a nearly optimal simulation of a high dimensional array with local access by a low dimensional one. The simulation is cast in terms of multidimensional Turing machines.

Introduced by Hartmanis and Stearns [2], multidimensional Turing machines are natural generalizations of conventional Turing machines. Hennie and Grigor'ev [1, 3] established a lower bound of $\Omega(T(n)^{1+1/d-1/e})$ on the time required by a multihead d-dimensional Turing machine to simulate an e-dimensional machine of time complexity $T(n)$ on-line. Our simulation nearly achieves this bound.

Theorem 1. *For all $d \geq 1$ and all $e > d$, every multihead e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line by a multihead d-dimensional Turing machine in time $O(T(n)^{1+1/d-1/e}(\log T(n))^{O(1)})$.*

For the case $d = 1$, Pippenger and Fischer [10] devised an optimal simulation that runs in time $O(T(n)^{2-1/e})$ on-line. Grigor'ev [1] described an on-line simulation in time $O(T(n)^{1+1/d})$ when $e = d + 1$; even in this special case, Theorem 1 provides a better upper bound. Also, Grigor'ev proved that every storage modification machine of time complexity $T(n)$ can be simulated on-line by a d-dimensional machine in time $O(T(n)^{1+1/(d-1)})$; since every multidimensional Turing machine can be simulated in real time by a storage modification machine [15], every e-dimensional machine can be simulated on-line by a d-dimensional machine in time $O(T(n)^{1+1/(d-1)})$. The time required by our simulation is smaller, however.

Monien [8] demonstrated that every nondeterministic multidimensional machine can be simulated by a nondeterministic one-dimensional machine in time $O(T(n)(\log T(n))^2)$. We consider only deterministic machines.

Paul, Seiferas, and Simon [9] established nonlinear lower bounds on the time required to simulate multidimensional machines on-line by machines with fewer worktape heads. Furthermore, for $d \geq 2$, they simulated multihead d-dimensional machines of time complexity $T(n)$ by d-dimensional machines with two worktape heads (on separate tapes) in time $O(T(n)^{1+1/d-\alpha})$, $\alpha = 1/(d^2(d-1)+d)$, on-line; they nearly attained their lower bound $\Omega(T(n)^{1+1/d-\gamma})$ for every $\gamma > 2/(d^2+d)$. In contrast, our d-dimensional simulator for Theorem 1 has more worktape heads than the e-dimensional machine that it simulates.

2. Definitions

Let us review definitions for multidimensional Turing machines. To each cell of a d-dimensional worktape assign in the usual way a d-tuple of integers called the *coordinates* of the cell. The coordinates of adjacent cells differ in just one component by ± 1. The *origin* is the cell whose coordinates are all zero. A *d-dimensional Turing machine* has a finite-state control, a read-only input tape, a write-only

output tape, and a finite number of d-dimensional worktapes, each of which has a finite number of heads. At each step the machine reads the symbols in the cells on which the input and worktape heads are positioned, writes symbols on these worktape cells and possibly on the output tape too, and shifts each worktape head in one of $2d+1$ possible directions—either to one of $2d$ adjacent cells or to the same cell. Initially, all worktape cells hold blanks, and every worktape head is positioned on the origin of its tape.

Fix integers $d \geq 1$ and $e > d$ and a finite alphabet Δ. To establish Theorem 1, it suffices to exhibit an on-line simulation of a particular e-dimensional machine E with worktape alphabet Δ by a d-dimensional machine D in time $O(n^{1+1/d-1/e}(\log n)^{O(1)})$ because, as described below, E epitomizes the primitive storage and retrieval operations on an e-dimensional worktape with one head. To simulate a multitape e-dimensional machine with one access head per worktape, a d-dimensional simulator treats each worktape like a separate machine E, though with different inputs. Furthermore, multihead e-dimensional worktapes can be replaced by several e-dimensional worktapes with one head each without time loss: Leong and Seiferas [6] proved that every d-dimensional Turing machine can be simulated in real time by a d-dimensional machine having just one head on each of its worktapes.

Machine E has one head on one worktape and operates in real time as follows. At each step it reads another input symbol, called a *command*, that has the form $\langle b, \delta \rangle$, where $b \in \Delta$, and δ is one of the $2e+1$ directions in which the worktape head can shift. Suppose E is in a configuration in which the cell y scanned by the worktape head contains b'. When E then reads the input symbol $\langle b, \delta \rangle$, it writes b on y, writes b' on its output tape, and shifts the worktape head in direction δ. Call symbols of Δ *responses*. Let Σ be the set of commands for E. Machine E defines a function from Σ^* to Δ^* that maps a string of commands into a string of responses of the same length. Machine D *simulates* E *on-line* in time $T(n)$ if it computes this function in time $T(n)$ on inputs of length n, and for every input, for every j, machine D produces the jth response before reading the $(j+1)$st command.

On a given input string of commands, when E has processed just the first τ commands, we say that E is *at time* τ. When D has processed only the first τ commands (and produced the first τ output responses), D is *at simulated time* τ.

Consider a string of n commands. For simplicity, we describe a simulation in which n is available off-line. The simulation can be converted routinely to an on-line simulation with time loss of only a constant factor [9]. In essence, for $n' = 1, 2, 4, 8, \ldots$, a modified simulator repeats the simulation *ab initio* using n' for the length of the input until $n' \geq n$. When $n' > 1$, an output manager prevents repetitious responses by writing only the $(n'/2+1)$st through n'th responses on the output tape. In addition, the modified simulator marks all cells that its worktape heads visit. Before embarking on the simulation with the next value of n', the modified simulator erases its tapes via a depth-first traversal of the marked cells. Because D runs in time $T_D(n') = \Omega(n')$ on inputs of length n', the modified simulator runs

in time

$$T_D(1)+T_D(2)+\cdots+T_D(2^{\lceil \log n \rceil})=O(T_D(n))$$

on inputs of length n.

On a d-dimensional worktape, a *box* is a set of cells that form a d-dimensional cube. The *volume* of a box is the number of cells in it. The *base cell* of a box is the cell whose coordinates are the smallest. Cell y is *at distance* s from cell z if the shortest rectilinear path from y to z has length s; equivalently, a worktape head requires exactly s steps to move from y to z. Cell y is *well within* box C of side s if it is at distance strictly greater than $s/3$ from every cell outside C. The *relative position* of a cell y with respect to a box C is the list of coordinates of y when the base cell of C is taken as the origin; if y is at distance s from the base cell of C, then its relative position can be specified by a binary string of length proportional to $\log s$. Write $x(h, C)$ for the cell at relative positive h with respect to box C. The *relative position* of a box in C is the relative position of its base cell with respect to C.

We present the simulation in two versions. Version I introduces the fundamental ideas. Version II employs more sophisticated techniques to achieve the time bound $O(n^{1-1/d-1/e}(\log n)^{O(1)})$.

3. Version I of the simulation

Without loss of generality, assume n is a power of 2. Set $L=\log_2 n$ and $s_L=n$, and for $i \leq L-1$, set $s_i=s_{i+1}/2$.

For each $i=0, 1, \ldots, L$, the *blocks at level* i are precisely the boxes B of side $3s_i$ on the worktape of E for which every component of the coordinates of the base cell of B is an integral multiple of s_i. See Fig. 1. The blocks at level $i-1$ that are contained in a block B at level i are the *subblocks* of B. Every block at level i has at most $(3s_i/s_{i-1})^e$ subblocks. This covering of the worktape of E enjoys two important properties:

(i) for every cell y, there are exactly 5^e blocks at level i that contain a cell at distance s_i or less from y;

(ii) for every cell y in a block B at level i, if y is at distance strictly greater than s_{i-1} from every cell outside B, then there is a subblock B' in B at level $i-1$ such that y is well within B'; the relative position of B' in B is easily calculated from the position of y.

Reischuk [11] employs a similar covering.

Fix a block B_L at level L such that the origin is well within B_L. On every input sequence of n commands the worktape head of E remains within B_L.

In D a *page at level* 0 is a box of volume at least $(3s_0)^e$; for $i>0$, a *page at level* i is a box that has a *mass store* and a *directory*. If P is a page at level i and P' is

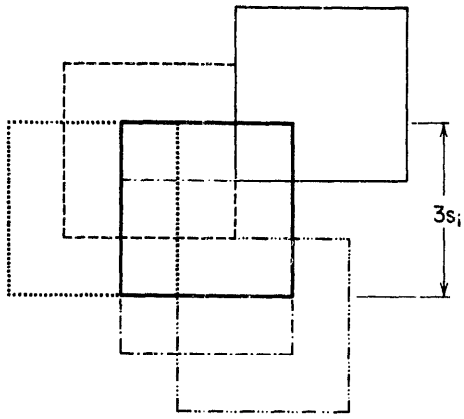

Fig. 1. Some blocks at level i.

a page at level $i-1$ and $P' \subseteq P$, then P' is *subpage* of P. We describe how the contents of a page P at level i *represent* the contents of a block B at level i.

If $i=0$, then P represents the contents of B literally: for each of the $(3s_0)^e$ cells y of B there is a representative cell z in P whose relative position in P is determined by the relative position of y in B, and z holds the same symbol as y. The details of this representation are unimportant, provided that relative position of z in P can be computed from the relative position of y in B in constant time.

If $i>0$, then for every nonblank subblock B' of B, there is a subpage of P whose contents represent the contents of B' recursively. All subpages at level $i-1$ are pairwise disjoint. Let P have side p. The mass store of P is a box of side $p/2$ in P that contains these subpages. The *address* of a box in the mass store is its relative position with respect to the mass store. The directory of P maintains the addresses of subpages of P. For Version I, the directory is a box of side $p/2$ that holds for every nonblank subblock B' its relative position in B together with the address of a subpage P' in P whose contents represent the contents of B'; subpage P' is *assigned* to B'.

Page P *represents block* B *at time* τ if its contents represent the contents of B in the configuration of E at time τ.

We present an informal overview of the simulation before giving the details. Suppose in a configuration of E the worktape head is well within block B at level i, and the contents of page P at level i represent the contents of B. Using P, procedure SIMULATE processes the next s_i commands in s_i/s_{i-1} phases of s_{i-1} commands each. At each phase, SIMULATE determines the subblock B' of B such that the worktape head of E is well within B'. Procedure PAGEADDRESS employs the directory of P to locate the subpage P' of P assigned to B'. Next, SIMULATE uses P' to process s_{i-1} commands recursively. Then for each subblock C' of B that meets B', procedure UPDATE recursively updates the contents of the subpage of P assigned to C' in order to represent the new contents of C'.

Set $u_0 = \lceil (3s_0)^{e/d} \rceil$, and for $i = 1, \ldots, L$, $u_i = 2\lceil (3s_i/s_{i-1})^{e/d} \rceil u_{i-1}$. In Version I, every page at level i has side u_i throughout the simulation. By induction on i, since a block at level i has at most $(3s_i/s_{i-1})^e$ subblocks at level $i-1$, the mass store of a page at level i, which has volume $(u_i/2)^d$, can hold $(3s_i/s_{i-1})^e$ subpages at level $i-1$ to represent the contents of all subblocks if they are all nonblank.

The simulator D has several heads on one d-dimensional worktape. To simulate E on an input string of n commands, move the worktape heads of D to the base cell of the page P_L of side u_L that represents B_L at time 0, and call SIMULATE with parameters (L, h_0), where h_0 is the relative position of the origin with respect to B_L.

Procedure $\sigma \leftarrow$ SIMULATE(i, h):

Hypothesis: At the beginning and end of this call to SIMULATE, all heads of D are on the base cell of a page P at level i.

Parameters: h is a binary string of length $O(\log s_i)$ that specifies the relative position of a cell in a block B at level i.

Value returned: σ is a sequence of s_i commands.

Effects: During this procedure call, the next s_i commands are read, the corresponding responses are produced, and the contents of P are changed. Suppose that at the beginning of this call, D is at simulated time τ, page P represents B at level i at time τ, and at time τ the worktape head of E is on $x(h, B)$, which is well within B. (Consequently, the worktape head of E is in B at times $\tau+1, \ldots, \tau+s_i$.) Then at the end of this call P represents B at time $\tau+s_i$, and σ is the sequence of commands at times $\tau+1, \ldots, \tau+s_i$.

Method: If $i = 0$, then simulate directly in P for s_0 steps: read s_0 commands and produce the appropriate responses. Let σ be the sequence of s_0 commands read.

Otherwise, if $i > 0$, then set $k \leftarrow h$ and perform Step 1 through Step 7 exactly s_i/s_{i-1} times, accumulating σ in Step 6.

Step 1. Use k to calculate the relative position of the subblock B' of B such that $x(k, B)$ is well within B'; property (ii) of the definition of the blocks asserts that B' exists. Calculate the relative position h' of $x(k, B)$ with respect to B'.

Step 2. Call PAGEADDRESS to retrieve the address of the subpage P' of P assigned to B'.

Step 3. Move the worktape heads of D to the base cell of P' and call SIMULATE $(i-1, h')$ to process the next s_{i-1} commands. Let σ' be the sequence of commands produced by this call.

Step 4. For each of the at most $5^e - 1$ subblocks C' of B such that $C' \neq B'$ and C' contains a cell at distance s_{i-1} or less from $x(k, B)$, perform Steps 4.1 and 4.2.

Step 4.1. Call PAGEADDRESS to determine the address of the subpage Q' of P assigned to C'. Let k' be the relative position of $x(k, B)$ with respect to C'.

Step 4.2. Move the heads of D to the base cell of Q' and call UPDATE $(i-1, k', \sigma')$.

Step 5. Use σ' to update k: set $k \leftarrow$ SHIFT (k, σ').

Step 6. Append σ' to σ.

Step 7. Return the heads of D to the base cell of P.

Correctness: To verify that SIMULATE operates properly, establish by induction that for each j, at the jth execution of Step 7, P represents B at time $\tau + js_{i-1}$ and the worktape head of E is on $x(k, B)$ at time $\tau + js_{i-1}$.

Procedure $h' \leftarrow$ SHIFT (h, σ):

Parameters: h is the relative position of a cell with respect to a box C on the worktape of E. σ is a sequence of commands.

Value returned: h' is the relative position of the head of E with respect to C that results from starting on cell $x(h, C)$ and performing the shifts indicated by σ.

Method: Using e unary counters, one for each dimension to maintain the displacement of the head from $x(h, C)$, change one of these counters by ± 1 for each of the shifts in σ. Finally, with e additions or subtractions, calculate the new relative position h'.

Procedure UPDATE (i, h, σ):

Hypothesis: At the beginning and end of this call to UPDATE, the worktape heads of D are on the base cell of a page Q at level i.

Parameters: h is a binary string of length $O(\log s_i)$ that specifies the relative position of a cell with respect to a block C at level i. σ is a sequence of s_i commands.

Effects: If at the beginning of this procedure call Q represents C at time τ, the worktape head of E is on $x(h, C)$ at time τ, and σ is the sequence of commands at times $\tau+1, \ldots, \tau+s_i$, then at the end of the call, Q represents C at time $\tau+s_i$.

Method: If $i=0$, then use σ to determine the new contents of every cell y in C that is visited by the worktape head of E when it starts from $x(h, C)$ and shifts according to σ; copy this new symbol into the representative of y in Q.

Otherwise, if $i>0$, then set $k \leftarrow h$; partition σ into s_i/s_{i-1} consecutive subsequences σ' of length s_{i-1}; and perform Step 1 through Step 3 for each σ', in order.

Step 1. For each of the at most 5^e subblocks C' of C that contains a cell within distance s_{i-1} of $x(k, C)$, perform Steps 1.1 and 1.2.

Step 1.1. Call PAGEADDRESS to determine the address of the subpage Q' of Q assigned to C'. Let h' be the relative position of $x(k, C)$ with respect to C'.

Step 1.2. Move the heads of D to the base cell of Q' and call UPDATE $(i-1, h', \sigma')$.

Step 2. Set $k \leftarrow$ SHIFT (k, σ').

Step 3. Return the heads of D to the base cell of Q.

Correctness: To check that UPDATE operates properly, show by induction that for each j, at the jth execution of Step 3, σ' is the sequence of commands at times $\tau+(j-1)s_{i-1}+1, \ldots, \tau+js_{i-1}$, the worktape head of E is on $x(k, C')$ at time $\tau+js_{i-1}$, and Q represents C at time $\tau+js_{i-1}$.

Procedure $a \leftarrow$ PAGEADDRESS (i, k'):

Hypothesis: The worktape heads of D are on a page P at level i.

Parameters: k' is a binary string of length $O(\log s_i)$ that specifies the relative position of a subblock B' of a block at level i.

Value returned: a is the address of a subpage P' in P assigned to B'.

Effects: This procedure may alter the contents of the directory and may set up a new page in the mass store.

Method: Using k' and the directory of P, retrieve the address of the subpage P' of P assigned to B'. If no address is associated with k', then find a blank box of side u_{i-1} in the mass store. Initialize this box so that it becomes a subpage P' whose contents represent a block whose cells all hold blanks. Return the address of P' in P as the value of a.

4. Version II of the simulation

Version I runs slowly because the worktape heads of D travel across large pages of fixed size that are mostly blank. Version II employs pages of different sizes: a small page can represent the contents of a block in which few cells are nonblank. In particular, the contents of a page at level L of volume only slightly larger than n represent the contents of the block B_L at level L at the beginning and the end of the computation on an input of length n. The efficiency of the simulation depends on the economy of these representations.

To decrease the simulation time further, we design a directory that permits rapid retrieval of subpage addresses. In a page of side p, to obtain the address of a subpage that represents a subblock, given the relative position of the subblock, takes time at most proportional to p.

Let π be the function defined by

$$\pi(x) = 2^{\lceil \log x \rceil};$$

if x is not a power of 2, then π maps x to the next larger power of 2. Set

$$c_0 = 8^d 5^{de}, \qquad \rho = e/d, \qquad s_L = \pi(n),$$

$$s_{L-1} = \pi(n^{1/e}/2), \qquad s_{L-2} = s_{L-1}/4,$$

$$s_{i-1} = \pi(s_i(s_i/s_{i+1})^\rho/4) \quad \text{for } i \leq L-2,$$

where L is the smallest integer such that $s_0 = 1$. If $\rho > 1$, then $L = O(\log \log n)$; if $\rho = 1$, then $L = O((\log n)^{1/2})$. Clearly, s_{i-1} divides s_i without remainder for every i. Let $m_L^* = n$, and for $i < L$ let

$$m_i^* = (3s_i)^e,$$

the volume of a block at level i. For all i set

$$\gamma_i = 3^e c_0^i, \qquad t_i = \pi((\gamma_i s_i)^{1/d}), \qquad u_i = \pi((\gamma_i m_i^*)^{1/d}).$$

Note that $u_L = O((c_0^L n)^{1/d}) = O(n^{1/d}(\log n)^{O(1)})$ if $\rho > 1$, and $u_L = O(n^{1/d + O((\log n)^{-1/2})})$ if $\rho = 1$. Furthermore, since $s_{i+1}^{\rho}/s_i \geq s_i^{\rho}/4s_{i-1}$ for all $i \leq L-2$,

$$u_{i+1}/s_i \geq 5^e u_i/s_{i-1} \quad \text{for } i \leq L-2. \tag{1}$$

Also, by definition,

$$u_L/s_{L-1} \geq 5^e u_{L-1}/s_{L-2}. \tag{2}$$

As before, for $i = 0, \ldots, L$, cover the worktape of E with overlapping boxes of side $3s_i$ called *blocks at level i*. The origin is well within block B_L at level L.

For Version II, a *page* at level i is a box on the worktape of D whose side is a power of 2. If $i > 0$, then it has a *free storage list* and a *nonblank cell counter* in addition to a mass store and a directory. We describe how the contents of a page P of side p at level i represent the contents of a block B at level i. If $i = 0$, then P uses a literal representation as in Version I; if $p = t_0$, then P is large enough to contain a representative cell for every cell of B. For $i > 0$, the mass store is a box of side $p/2$ in P. For every nonblank subblock B' of B there is a subpage of P at level $i-1$ in its mass store whose contents represent the contents of B' recursively. The subpages of P are pairwise disjoint.

The directory of P is a box of side $p/(4 \log s_i)$ in P. To maintain the addresses of subpages of P whose contents represent the contents of subblocks of B, the contents of the directory are organized into a binary trie [5] that uses the relative positions of subblocks as keys. More precisely, the relative positions of subblocks in B are specified by binary strings of length $O(\log s_i)$. Each binary string corresponds in a natural way to a leaf of a binary-unary tree of height $O(\log s_i)$. (Internal nodes of the tree have a left son or a right son or both.) The number of leaves of this tree is the number of subblocks of B whose contents are represented by the contents of subpages of P. See Fig. 2. We embed this tree, called the *directory trie*, in the directory in a straightforward manner. For every internal node ν of the trie, there is a *pointer box* in the directory that contains the relative positions of the pointer boxes for the one or two sons of ν. If B' is a nonblank subblock of B whose relative position in B corresponds to leaf node λ of the trie, then the pointer box for λ contains the address of the subpage P' of P whose contents represent the contents of B'. We say that P' is *assigned* to B' and that the directory *associates* an address with a relative position in B. Each pointer box in the directory of P has volume $O(\log u_i)$; it is sufficiently large to contain the relative positions of two other pointer boxes in the directory.

The free storage list is a list of addresses that occupies a box of side $p/4$ in P. For $q = 1, 2, 4, \ldots, p/4, p/2$, the free storage list has addresses of at most $2^d - 1$ blank boxes of side q in the mass store of P.

The nonblank cell counter specifies an integer between 0 and m_i^* (inclusive). This value of the nonblank cell counter is at least as large as the number of nonblank cells of B.

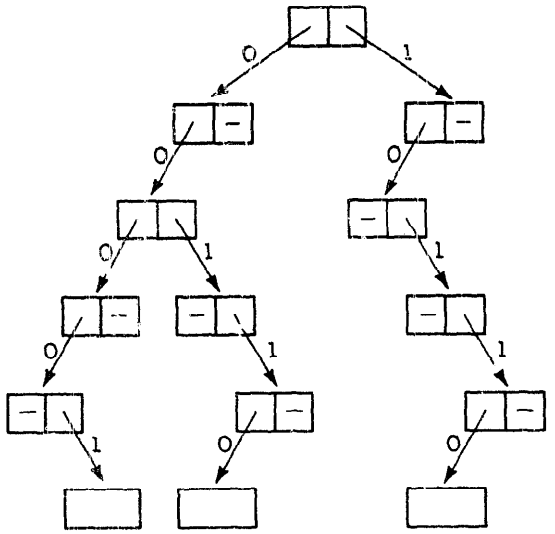

Fig. 2. The structure of a directory trie for a page that represents a block B with at most three nonblank subblocks. The relative positions of these subblocks in B are specified by the binary strings 00001, 00110, 10110.

Throughout Version II of the simulation, the value of the nonblank cell counter of a page determines its size. If the nonblank cell counter has value m, then the page has side $\pi((\gamma_i m)^{1/d})$. Since $m \leq m_i^*$, every page at level i has side at most u_i.

Procedures SIMULATE and UPDATE operate as in Version I, though they use the values of s_i defined for Version II. Procedure PAGEADDRESS is substantially different.

Procedure $a \leftarrow$ PAGEADDRESS(i, k'):

Hypothesis: The worktape heads of D are on a page P at level i.

Parameters: k' is a binary string of length $O(\log s_i)$ that specifies the relative position of a subblock B' of a block at level i.

Value returned: a is the address of a subpage P' in P assigned to B' such that the side of P' is $\pi((\gamma_{i-1}m')^{1/d})$, where m' is the value of the nonblank cell counter of P'.

Effects: This procedure changes the value of the nonblank cell counter of P', may alter the contents of the directory and the free storage list by a call to ALLOCATE, and may set up a new page in the mass store.

Method: Using k' and the directory of P, retrieve the address of the subpage P' of P assigned to B': visit the $O(\log s_i)$ pointer boxes for the nodes on the path in the directory trie from the root to the leaf that corresponds to k' to obtain the address associated with k'.

If no address is associated with k', then call ALLOCATE (i, t_{i-1}, k') to obtain a blank box of side t_{i-1} in the mass store. Initialize this box so that it becomes a subpage P' whose contents represent a block whose cells all hold blanks: the free storage list of P' contains just the address of the blank box of side $t_{i-1}/2$ in its

mass store (namely, the mass store itself); the value of the nonblank cell counter of P' is s_{i-1}. Return the address of P' in P as the value of a.

Let p' be the side of P' and m' be the value of its nonblank cell counter; by definition, p' is a power of 2. Set the value of the nonblank cell counter of P' to $m'' = \min\{m^*_{i-1}, m' + s_{i-1}\}$. Let $p'' = \pi((\gamma_{i-1} m'')^{1/d})$. If $p' < p''$, then call ALLOCATE(i, p'', k') to obtain a new box of side $p'' \geq 2p'$ in the mass store of P assigned to B'. Copy the contents of P' into this box to produce a larger page P'' such that if the contents of P' represent the contents of B', then the contents of P'' also represent the contents of B'; in particular, to ensure that the addresses in the directory remain valid, copy the contents of the mass store of P', which has side $p'/2$, into the box of side $p'/2$ whose base cell is the same as the base cell of the mass store of P''. Copy the free storage list of P' into the free storage list of P''. On the free storage list of P'' enter the addresses of $2^d - 1$ blank boxes of side $p''/4$ in its mass store. In this case return the address of P'' in P as the value of a.

Procedure $a \leftarrow$ ALLOCATE (i, r, k'):

Hypothesis: The worktape heads of D are on a page P of side p at level i.

Parameters: k' is a binary string of length $O(\log s_i)$. r is a power of 2.

Value returned: a is the address of a blank box of side r in the mass store of P, if this procedure call does not fail.

Effects: The contents of the directory are altered to associate a with k', and the free storage list of P is changed. (During this call to ALLOCATE, the free storage list may temporarily hold addresses of 2^d blank boxes of the same side.)

Method: A buddy system is used [4].

Step 1. If the free storage list has an address of a box of side r, then skip to Step 2. Let q^* be the smallest power of 2 greater than r for which the free storage list does have an address of a box of side q^*; if no such q^* exists, then terminate with failure. For $q = q^*, q^*/2, \ldots, 4r, 2r$ in order, select an address a_q of a box C_q of side q, delete a_q from the list, and add to the list the addresses of the 2^d pairwise disjoint boxes of side $q/2$ whose union is C_q. The free storage list now has addresses of $2^d - 1$ blank boxes of sides $2r, 4r, \ldots, q^*/2$ and of 2^d blank boxes of side r.

Step 2. Let a be the address of a box of side r on the free storage list, and delete this address from the list. In the directory of P, set up at most $O(\log s_i)$ pointer boxes of volume $O(\log u_i)$ for the directory trie to associate address a with k'. If the pointer boxes for the directory trie no longer fit in a box of side $p/(4 \log s_i)$, then terminate with failure.

To simulate E on an input string of n commands, move the worktape heads of D to the base cell of the page P_L of side u_L that represents B_L at time 0, and call SIMULATE with parameters (L, h_0), where h_0 is the relative position of the origin with respect to B_L. Procedure ALLOCATE includes a provision for failure of the simulation. In Section 6 we prove that every call to ALLOCATE in the simulation terminates successfully.

5. Time analysis of Version II

In analyzing the time used by Version II, we must account for the copying of subpages in calls to PAGEADDRESS. Since copying operations occur intermittently, we assess them separately.

In the directory of a page at level i of side $p \leq u_i$, to move a head from one pointer box to another takes time proportional to $p/(4 \log s_i)$, the side of the directory. Thus, to determine the address of the subpage assigned to a subblock or to associate an address with the relative position of a subblock takes time

$$O(\log s_i)[O(\log u_i) + O(p/(4 \log s_i))] = O(u_i)$$

because $O(\log s_i)$ pointer boxes, each of volume $O(\log u_i)$, are accessed.

Let $T_A(i)$ be the time used by ALLOCATE on a page at level i. Since time $O(u_i)$ is consumed in moving the heads around the page in the directory and time $O((\log u_i)^2)$ in handling the addresses in the free storage list,

$$T_A(i) = O(u_i) + O((\log u_i)^2) = O(u_i).$$

Let $T_P(i)$ be the time consumed by PAGEADDRESS on a page level i, *excluding the copying of subpages.* This procedure retrieves an address from the directory (time $O(u_i)$), performs some arithmetic calculations (time $O(\log u_i)$), moves heads around the mass store (time $O(u_i)$), and calls ALLOCATE:

$$T_P(i) = O(\log u_i) + O(u_i) + T_A(i) = O(u_i).$$

Procedure SHIFT operates in time proportional to the sum of the lengths of its inputs.

Let $T_U(i)$ be the time used by UPDATE on pages Q at level $i < L$, excluding the copying of subpages in calls to PAGEADDRESS. Evidently, $T_U(0) = O(1)$. For $i > 0$ we assess the time taken by each step:

Step 1.1: $O(\log s_i) + T_P(i)$ for each iteration of Step 1—$O(\log s_i)$ for calculations and $T_P(i)$ for the call to PAGEADDRESS.

Step 1.2: $O(u_i) + T_U(i-1)$ for each of at most 5^e iterations of Step 1—$O(u_i)$ to move the heads across Q, whose side is at most u_i, and $T_U(i-1)$ for the recursive call.

Step 2: $O(\log s_i) + O(s_{i-1})$ for the call to SHIFT for each iteration.

Step 3: $O(u_i)$ for each iteration.

Thus, by induction on i, (1), and (2), there exists a constant k_1 such that

$$\begin{aligned} T_U(i) &\leq (s_i/s_{i-1})[O(\log s_i) + O(s_{i-1}) + O(u_i) + O(T_P(i)) + 5^e T_U(i-1)] \\ &\leq (s_i/s_{i-1})[O(u_i) + 5^e T_U(i-1)] \\ &\leq (s_i/s_{i-1})[k_1 u_i + 5^e k_1 i s_{i-1} u_{i-1}/s_{i-2}] \\ &\leq k_1 s_i (u_i/s_{i-1} + 5^e i u_{i-1}/s_{i-2}) \\ &\leq k_1 (i+1) s_i u_i / s_{i-1}. \end{aligned}$$

Let SIMULATE run within time $T_S(i)$ on every page at level i, excluding the time for the copying of subpages in PAGEADDRESS. Clearly, $T_S(0) = O(1)$. For $i > 0$ we reckon the time used by each step:

Step 1: $O(\log s_i)$ for each iteration.

Step 2: $T_P(i)$ for each iteration.

Step 3: $O(u_i) + T_S(i-1)$ for each iteration—$O(u_i)$ to move the heads and $T_S(i-1)$ for the recursive call.

Step 4.1: $O(\log s_i) + T_P(i)$ for each iteration of Step 4.

Step 4.2: $O(u_i) + T_U(i-1)$ for each of at most $5^e - 1$ iterations of Step 4.

Step 5: $O(\log s_i) + O(s_{i-1})$ for each iteration.

Step 6: $O(s_{i-1})$ for each iteration.

Step 7: $O(u_i)$ for each iteration.

Then by induction, for another constant $k_2 \geq k_1$,

$$\begin{aligned} T_S(i) &\leq (s_i/s_{i-1})[O(\log s_i) + O(s_{i-1}) + O(u_i) + O(T_P(i)) + (5^e - 1)T_U(i-1) \\ &\qquad + T_S(i-1)] \\ &\leq (s_i/s_{i-1})[O(u_i) + (5^e - 1)T_U(i-1) + T_S(i-1)] \\ &\leq (s_i/s_{i-1})[k_2 u_i + (5^e - 1)k_1 i s_{i-1} u_{i-1}/s_{i-2} + k_2 i s_{i-1} u_{i-1}/s_{i-2}] \\ &\leq k_2 s_i (u_i/s_{i-1} + 5^e i u_{i-1}/s_{i-2}) \\ &\leq k_2 (i+1) s_i u_i / s_{i-1}. \end{aligned}$$

We prove that during the simulation of E by D, the total time taken by copying the contents of pages in calls to procedure PAGEADDRESS is $O(u_L^d)$. Let page P have volume v. Let $Q_1, Q_2, \ldots, Q_f = P$ be the sequence of pages such that during the simulation, for each $j > 1$, PAGEADDRESS called ALLOCATE to obtain Q_j and copied the contents of Q_{j-1} into Q_j. Call the sum over j of the time for copying Q_{j-1} into Q_j the *ancestral copying time* for P. The sides of these Q_j are increasing powers of 2. Consequently, since the time to copy the contents of Q_{j-1} into Q_j is bounded above by the volume of Q_j, the ancestral copying time for P is at most twice the volume of P, namely, $2v$. Let $P_1, P_2, \ldots$ be the subpages of P (at level $i-1$) and $v_1, v_2, \ldots$ be their volumes. Since these pairwise disjoint subpages lie in the mass store of P, whose volume is $v/2^d$,

$$\sum_j v_j \leq v/2^d.$$

Suppose inductively that for each j, the sum of the ancestral copying times for P_j and its subpages at all levels is at most $4v_j$. Then the sum of the ancestral copying times for P and its subpages at all levels is at most

$$2v + \sum_j 4v_j \leq 2v + 4v/2^d \leq 4v.$$

In particular, when $P = P_L$, the page at level L assigned to B_L, this total copying time is at most $4u_L^d$.

Therefore, the simulation uses time

$$T_S(L)+4u_L^d \leq k_2(L+1)s_L u_L / s_{L-1}+4u_L^d$$
$$= \begin{cases} O(n^{1+1/d-1/e}(\log n)^{O(1)}) & \text{if } \rho > 1, \\ O(n^{1+1/d-1/e+O((\log n)^{-1/2})}) & \text{if } \rho = 1. \end{cases}$$

The heads of D remain within distance u_L of the origin. Assuming the successful termination of the simulation (established in Section 6), we summarize these results in a form that subsumes Theorem 1.

Theorem 2. *For all $d \geq 1$ and all $e > d$, every multihead e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line in time $O(T(n)^{1+1/d-1/e}(\log T(n))^{O(1)})$ by a multihead d-dimensional Turing machine whose heads remain within distance $O(T(n)^{1/d}(\log T(n))^{O(1)})$ of the origin. Every multihead d-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line in time $O(T(n)^{1+O((\log T(n))^{-1/2})})$ by a multihead d-dimensional Turing machine whose heads remain within distance $O(T(n)^{1/d+O((\log T(n))^{-1/2})})$ of the origin.*

6. Successful termination of Version II

We prove that every call to ALLOCATE completes without failure.

In a configuration of D, the *free volume* of a page is the total volume of the boxes whose addresses are on the free storage list.

Lemma 1. *Let r be a power of 2. Suppose the free volume of page P is at least r^d in a configuration of D at the beginning of a call to* ALLOCATE *on P. Then this call can produce the address of a blank box of side r in the mass store of P.*

Proof. Let $q_1 \leq q_2 \leq \cdots \leq q_k$ be the sides of boxes whose addresses are on the free storage list of P. Since the free storage list has at most $2^d - 1$ boxes of each distinct side, the free volume v of P satisfies

$$v = q_k^d + \cdots + q_1^d \leq (2^d-1)q_k^d + (2^d-1)(q_k/2)^d + \cdots + (2^d-1)(1)^d < (2q_k)^d.$$

If $r^d \leq v$, then $r^d < (2q_k)^d$, hence since r is a power of 2, $r \leq q_k$. Consequently, ALLOCATE can find a blank box of side r in the mass store of P. □

Let D be at simulated time τ at the beginning of a call to SIMULATE on a page P at level $i > 0$ that represents block B at time τ. Let m be the value of the nonblank cell counter of P in this configuration. Call a subpage of P *active* if it is assigned to some subblock of B. We show that during this call to SIMULATE, every call to ALLOCATE made by a call to PAGEADDRESS in Step 2 or Step 4.1 completes successfully. The argument for calls to ALLOCATE during a call to UPDATE is identical.

Let R be the page such that P is in the mass store of R. Observe that the invocation of SIMULATE on R making the recursive call on P calls PAGEADDRESS on R first to locate P and to increment its nonblank cell counter to m.

Lemma 2. *Throughout this call to* SIMULATE

(i) *P has at most $5^e m/s_{i-1}$ active subpages, and*

(ii) *the total of the nonblank cell counters of the active subpages of P never exceeds $5^e m$.*

Proof. First, suppose $m = m_i^*$. Since B has at most $(3s_i/s_{i-1})^e$ subblocks, P has at most $(3s_i/s_{i-1})^e \leq m_i^*/s_{i-1} \leq 5^e m/s_{i-1}$ active subpages, and the sum of their nonblank cell counters is at most

$$(3s_i/s_{i-1})^e m_{i-1}^* = 3^e m_i^* \leq 5^e m.$$

Now suppose PAGEADDRESS incremented the nonblank cell counter of P from $m - s_i$ to m just before the call to SIMULATE. By induction on τ, at simulated time τ, P has at most $5^e(m - s_i)/s_{i-1}$ active subpages, and the total of their nonblank cell counters is at most $5^e(m - s_i)$. At each of s_i/s_{i-1} iterations during the execution of SIMULATE, at most 5^e new active subpages are created, and s_{i-1} is added to the nonblank cell counters of at most 5^e subpages. Thus, the number of active subpages of P is always at most

$$5^e(m - s_i)/s_{i-1} + 5^e(s_i/s_{i-1}) = 5^e m/s_{i-1}.$$

The total of the nonblank cell counters of the active subpages of P is at most

$$5^e(m - s_i) + 5^e(s_i/s_{i-1})s_{i-1} = 5^e m \quad \square$$

We verify that P has enough space for the free storage list and for the pointer boxes in the directory during this call to SIMULATE. Since the side of P is $\pi((\gamma_i m)^{1/d}) \leq \pi((\gamma_i m_i^*)^{1/d}) = u_i$, the relative position of a box in P can be specified by a string of $O(\log u_i)$ symbols. (By choosing the size of the worktape alphabet of D, we can adjust the constant of proportionality to ensure that assertions (3) and (4) below are true.) The free storage list of P comprises $O(\log u_i)$ addresses of length $O(\log u_i)$; thus, by definition of u_i and t_i, the free storage list occupies a box of side

$$O((\log u_i)^{2/d}) \leq O(t_i^{1/d}) \leq \pi((\gamma_i m)^{1/d})/4 \tag{3}$$

because $m \geq s_i$. In the directory of P there are $O(\log s_i)$ pointer boxes of fixed volume $O(\log u_i)$ for each of the $O(m/s_{i-1})$ active subpages of P. These pointer boxes fit in a box of volume

$$O((m/s_{i-1})(\log s_i)(\log u_i)) \leq (\pi((\gamma_i m)^{1/d})/(4 \log s_i))^d, \tag{4}$$

the volume of the directory of P. When PAGEADDRESS calls ALLOCATE on P, this call cannot fail for lack of space for the directory.

Consider a configuration of D just before a call to PAGEADDRESS on P in Step 2 or Step 4.1 between simulated time τ and simulated time $\tau + s_i$. In this configuration let $P_1, P_2, \ldots$ be the active subpages of P and let $m_1, m_2, \ldots$ be the values of their nonblank cell counters; let P_j represent subblock B_j in B. The side of P_j is $\pi((\gamma_{i-1}m_j)^{1/d})$. The mass store of P holds the contents of smaller subpages that were assigned to B_j in previous configurations. Because the sides of these smaller subpages are distinct powers of 2, their total volume is at most the volume of P_j, namely, $(\pi((\gamma_{i-1}m_j)^{1/d}))^d$. Consequently, the volume of used boxes in the mass store of P in this configuration is bounded by

$$\sum_j 2(\pi((\gamma_{i-1}m_j)^{1/d}))^d.$$

Suppose PAGEADDRESS sets $m_1' = m_1 + s_{i-1}$ and decides to assign to B_1 a new larger page of side $\pi((\gamma_{i-1}m_1')^{1/d})$; according to the definition of PAGEADDRESS,

$$\pi((\gamma_{i-1}m_1')^{1/d}) \geq 2\pi((\gamma_{i-1}m_1)^{1/d}). \tag{5}$$

Lemma 2(ii) implies that

$$m_1' + \sum_{j>1} m_j \leq 5^e m. \tag{6}$$

Because the mass store has side $\pi((\gamma_i m')^{1/d})/2$, the free volume of P in this configuration is at least

$$\begin{aligned}
&(\pi((\gamma_i m)^{1/d})/2)^d - \sum_j 2(\pi((\gamma_{i-1}m_j)^{1/d}))^d \\
&\quad \geq 4^d 5^{de}\gamma_{i-1}m - 2(\pi((\gamma_{i-1}m_1')^{1/d})/2)^d - 2\sum_{j>1}(2^d\gamma_{i-1}m_j) \quad \text{by (5)} \\
&\quad \geq 4^d\gamma_{i-1}(5^e m) - 4^d\gamma_{i-1}\left(\sum_{j>1} m_j\right) - 2\gamma_{i-1}m_1' \\
&\quad \geq 4^d\gamma_{i-1}m_1' - 2\gamma_{i-1}m_1' \quad \text{by (6)} \\
&\quad \geq (\pi((\gamma_{i-1}m_1')^{1/d}))^d.
\end{aligned}$$

Lemma 1 guarantees that ALLOCATE can find a blank box of side $\pi((\gamma_{i-1}m_1')^{1/d})$ in the mass store of P. Therefore, this call to ALLOCATE completes successfully.

Acknowledgment

Discussions with Larry Carter, Dexter Kozen, and Robert Melville lead to improved upper bounds. Albert Meyer made detailed recommendations that simplified and clarified the description of the simulation. Harold Abelson and Neil Immerman also contributed expository suggestions. Joel Seiferas provided the reference to Monien [8].

References

[1] D.Ju. Grigor'ev, Imbedding theorems for Turing machines of different dimensions and Kolmogorov's algorithms, *Soviet Math. Dokl.* **18** (1977) 588–592.

[2] J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965) 285–306.

[3] F.C. Hennie, On-line Turing machine computations, *IEEE Trans. Elec. Comput.* **15** (1966) 35–44.

[4] D.E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (Addison-Wesley, Reading, MA, 1968).

[5] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching* (Addison-Wesley, Reading, MA, 1973).

[6] B. Leong and J. Seiferas, New real-time simulations of multihead tape units, *J. ACM* **28** (1981) 166–180.

[7] R.J. Lipton, S.C. Eisenstat and R.A. DeMillo, Space and time hierarchies for classes of control structures and data structures, *J. ACM* **23** (1976) 720–732.

[8] B. Monien, About the derivation languages of grammars and machines, *Proc. 4th Colloquium on Automata, Languages and Programming* (Springer, Berlin, 1977) 337–351.

[9] W.J. Paul, J.I. Seiferas and J. Simon, An information-theoretic approach to time bounds for on-line computation, *J. Comput. System Sci.* **23** (1981) 108–126.

[10] N. Pippenger and M.J. Fischer, Relations among complexity measures, *J. ACM* **26** (1979) 361–381.

[11] R. Reischuk, A fast implementation of a multidimensional storage into a tree storage, *Proc. 7th International Colloquium on Automata, Languages, and Programming* (Springer, Berlin, 1980) 531–542.

[12] A.L. Rosenberg, Preserving proximity in arrays, *SIAM J. Comput.* **4** (1975) 443–460.

[13] A.L. Rosenberg, Data encodings and their costs, *Acta Informat.* **9** (1978) 273–292.

[14] A.L. Rosenberg and L. Snyder, Bounds on the costs of data encodings, *Math. Systems Theory* **12** (1978) 9–39.

[15] A. Schönhage, Storage modification machines, *SIAM J. Comput.* **9** (1980) 490–508.

[16] H.-J. Stoss, Zwei-band Simulation von Turing Maschinen, *Computing* **7** (1971) 222–235.